KitFox: Multi-Physics Libraries for Integrated Power, Thermal, and Reliability Simulations of Multicore Microarchitecture

William J. Song, Saibal Mukhopadhyay, *Senior Member, IEEE*, and Sudhakar Yalamanchili, *Fellow, IEEE*

School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332
wjhsong@gatech.edu, saibal@ece.gatech.edu, sudha@gatech.edu

***Abstract*— With continued technology scaling and increased power and thermal densities, processor operation and performance are increasingly dominated by physical phenomena. Microarchitectural approaches to mitigate these effects must be based on a profound understanding of how the physics is manifested in microarchitectural executions and system-level properties such as performance, energy efficiency, or lifetime reliability. This requires a modeling and simulation environment that incorporates multiple physical phenomena and their concurrent interactions with microarchitecture. In this paper, we introduce an integrated power, thermal, and reliability modeling framework, *KitFox*. The goal of KitFox framework is to facilitate research explorations at the intersection of applications, microarchitectures, and various physical phenomena including energy, power, thermal, cooling, and reliability. The KitFox framework implements a standard interface to bridge multiple physical models, where individual models are encapsulated into libraries and are interchangeable. This paper describes the design methodology of the library framework that orchestrates various implementations of physical models and standardized interface to cycle-level microarchitecture simulators. Several use cases are presented to demonstrate the range of modeling capabilities of KitFox.**

I. Introduction

Microarchitectural operations are limited by physical challenges. Dynamic control schemes such as power or thermal management combined with inherent workload dynamics create spatiotemporal changes in thermal and voltage states. These in turn lead to variations in leakage power, logic delay, or device degradation across a processor die, which eventually impact system-level properties such as performance, energy efficiency, and lifetime reliability. Modeling and simulation of future microarchitectures and applications must be *holistic* including power, energy, thermal, and reliability concerns in that their coupled interactions are becoming major performance determinants of processors. Hence, we require more sophisticated modeling and simulation infrastructures that combine traditional multicore timing simulation with interacting physical models.

We note that an integrated modeling and simulation infrastructure must depart from the state of the practice to accurately capture interactions between multiple physical phenomena and their impact on microarchitectures. For instance, trace-driven simulations have been widely used, where distinct physical properties are calculated in the independent steps of simulations. Such approaches do not model interdependency between physical phenomena such as temperature and leakage power. Even if such feedback is implemented (usually in a tool-specific manner), it cannot be correctly reflected in the entire chain of models. Ideally, we need a modeling infrastructure where interactions between all physical phenomena can be modeled and driven by applications executing on a target multicore microarchitecture; modeling of integrated application, microarchitecture, and physics combinations encompassing all interactions and feedback loops. We refer to the modeling of interactions between diverse physical phenomena as *multi-physics modeling* in this paper. To address this important modeling problem, we have been developing *KitFox*, an open-source modeling framework (formerly called Energy Introspector [20]). We have integrated KitFox with different multicore simulation infrastructures including Manifold [27], Structural Simulation Toolkit (SST) [6], [16], and MacSim [12] and applied it to a range of research problems requiring integrated multi-physics and multicore simulations [1], [2], [12], [15], [21], [22], [29].

In this paper, we discuss the design methodology, implementation, and usage of KitFox. Architecture-level multi-physics simulations utilize various physical models. In particular, KitFox is based on the integration of external models that simulate different physical properties; reliability, power, and thermal including popular tools in the architecture community such as HotSpot [7] and McPAT [11]. Each type of model in KitFox is encapsulated into a standard class called *library* (e.g., energy, thermal, or reliability library). Any new or updated models can be seamlessly integrated into KitFox by encapsulating them into the respective libraries. There are no hidden software dependencies between libraries, and all interactions are explicitly managed via a standard library interface to avoid modifications to the third party tools. Microarchitecture simulations invoke KitFox through user application programming interface (API), which relieves microarchitecture modelers from orchestrating all multi-physics interactions. Microarchitecture simulators themselves do not need to incorporate and handle the physical models. Figure 1 illustrates the concept of standard libraries and user API. KitFox has the following features and contributions:

- *Standard library*: Each model is encapsulated into a standard class called *library* and integrated to KitFox. No cross-dependency in software integration is created between models, and any new models can be seamlessly added to the framework.
- *Interacting library models*: Interactions between physical models are orchestrated inside KitFox via a standard library interface. It minimizes user involvement in data management to coordinate multiple libraries and subclass models.
- *Simple user API*: KitFox provides a set of user API functions to be used in microarchitecture simulations. The same function is used for calculating the same physical property via the standard library interface, regardless of which individual model is used inside the framework. For instance, all thermal models can be accessed the same way from microarchitecture simulators.
- *Configurable processor model*: KitFox provides a configuration method that users can define the microarchitectural and physical hierarchy of a processor package and associate individual physical models with constituent processor components to be simulated.
- *Parallel simulation via MPI interface*: KitFox supports parallel simulation of physical models via message passing interface (MPI) implementations. A microarchitecture simulator and KitFox can execute in parallel in separate MPI processes, and KitFox itself can also be divided into multiple MPI ranks.

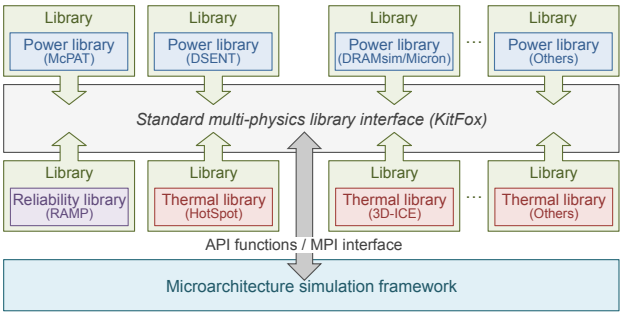


Figure 1. Multiple physical models are integrated into KitFox as *libraries*. It standardizes the integration and use of models, and it provides microarchitecture simulators with a set of API functions to invoke the models [7], [11], [17], [23], [24], [25].

The remainder of the paper is organized as follows. We first explain our motivation for developing KitFox and distinctions from related efforts. We outline the process of building a full-system simulation environment including applications, microarchitectures, and various physical models. Then, we detail the methodology of architecture-level multi-physics simulations based on the proposed standard library approach. In particular, we describe how multiple physical models are integrated into KitFox and how their interactions are orchestrated. Various engineering problems encountered in the development and solutions we developed are discussed, such as data manipulation and time synchronization. Finally, we present several use cases of KitFox to demonstrate the versatility of the framework.

II. Design Motivation and Distinction From Previous Work

We are motivated by two major issues. First, we note that there already exist a variety of popular point tools used in the architecture community and also many other custom models in industry. Considerable efforts have been already invested to develop, validate, and release these models. Utilizing them is a pragmatic, cost-effective, and convenient start rather than re-investing resources and time to develop new models with similar capabilities. Furthermore, the architecture research community continues to develop new models or update existing tools as technology evolves. Therefore, it is desirable for a modeling framework to support the integration of various implementations of tools and be open to the easy integration of new models as they are developed and become available.

The second challenge is the efficient integration of these models with microarchitecture simulators while supporting multi-physics interactions between them. To date, this has normally been a tedious, laborious, and error-prone engineering effort. We argue that standardization of model interface is a logical approach and should take the form of an API that is standardized across model types and invocations. Such a framework is portable across multicore simulation infrastructures with only engineering efforts for porting the API rather than re-integrating physical models into each detailed microarchitecture simulator.

There are several related efforts towards the development of a multi-physics simulation environment. Coskun et al. [5] presented a trace-driven simulation method including power, thermal, and reliability models. Traces are the easiest way of connecting disparate simulation tools, but this approach is fundamentally limited in its ability to capture runtime dynamics. For example, a runtime control via dynamic frequency scaling (DFS) significantly changes time-sampled data in traces. It requires artificial post-processing of pre-generated traces to reflect dynamic changes, which is difficult to perform accurately and is error-prone. In contrast, KitFox takes a more holistic approach by integrating various models into the same framework. It provides a simple and standardized API to microarchitecture simulators to facilitate the use of integrated models.

Bartolini et al. [4] introduced a MATLAB-based infrastructure interfaced with a conventional C/C++-written microarchitecture simulator via copying of data structures to/from shared memory space. Although MATLAB is a powerful toolkit, it is not suitable to be used in the direct integration of already time-consuming microarchitecture simulations. The authors implemented empirically obtained physical models in MATLAB, whereas KitFox pursues support of various validated, open-source modeling tools that are written in C/C++ and popularly used in the research community.

Sajjadi-Kia and Ababei [18] developed a thermal-reliability simulation framework to optimize floorplanning. Their framework used the detailed circuit-level information of an IP block from a SPICE simulation. The authors targeted at exploring various floorplanning options by varying design parameters, which was conducted in a static manner at each step of iterative simulation. Priyadarshi et al. [14] presented a simulation framework for the thermal pathfinding of 3D ICs. The authors used simplified system description instead of using detailed and therefore slow CAD models. Their framework also aimed at implementing a metric-driven tool flow to find an optimal 3D design by changing design parameters. KitFox framework is distinct from these efforts in that it models workload-driven runtime dynamics in microarchitecture operations and multi-physics interactions. It also pursues flexibility in system description that enables users to compose different processor designs instead of enforcing a particular system such as 3D IC.

Developing an integrated multi-physics simulation environment involves numerous software engineering challenges to implement a standardized, configurable, and scalable simulation framework. In this paper, we introduce a novel multi-physics simulation framework, *KitFox*. The goal of KitFox framework is to provide a standardized integration method and interface to bridge various implementations of physical models and facilitate research explorations at the intersection of application, microarchitecture, power, energy, thermal, cooling, and reliability. We will also point out later other advantages including the use of interchangeable physical models with no modifications to simulators. Such features and capabilities are distinguished from the state of the practice that incorporates specific point tools via tight integration with simulators [4], [5], [14], [18]. To the best of our knowledge, there does not exist a microarchitecture-level multi-physics modeling framework with modularized models.

III. Library Integration of Physical Models

KitFox framework provides a standardized method to integrate various implementations of physical modeling tools. An individual tool in general is specialized to model a single type of physical property, and different tools have different functionalities and usages. In KitFox, we define classes called *libraries*, where each library hosts different type of models; *energy*, *thermal*, and *reliability libraries*. For each model being integrated into KitFox, a *wrapper class* is created. The wrapper class is defined as the subclass of one of the standard library classes listed in Table I. It includes header/source files of the tool to be integrated and re-defines the usage of the model according to the virtual functions of the corresponding library class. As a result, models of the same library type can be used in an identical way. For instance, HotSpot [7] and 3D-ICE [23] are popular tools used for package-level thermal modeling. Both models are integrated into the thermal library in KitFox and driven by the same functions, although their implementations are different. When KitFox is used in a microarchitecture simulator, the choice of one

TABLE I. Standard Library Classes in KitFox

Library types	Description
Energy library	• Estimation of per-access energies of different access types (e.g., read, write, logical switching) • Area estimation based on circuit-level models • Runtime updates of variables (e.g., voltage, clock frequency, temperature) • Integrated models: McPAT (including Cacti) [11], [26], DSENT (previously named Orion) [9], [25], DRAMSim (Micron model) [17], IntSim [19]
Thermal library	• Floor-planning and power grid mapping • Calculation of steady-state or transient temperatures • Runtime updates of variables (e.g., ambient temperature, coolant flow rate, power distribution) • Integrated models: HotSpot [7], 3D-ICE [23]
Reliability library	• Calculation of cumulative failure rates • Runtime updates of variables (e.g., temperature) • Integrated models: RAMP-like failure models including electro-migration (EM), bias temperature instability (BTI), time-dependent dielectric breakdown (TDDB), hot carrier injection (HCI), stress migration (SM) [8], [24], [28]

or the other model can be made with no changes to the simulator; model selection is specified in an input configuration file. Table I summarizes the functions of library classes.

A. Energy Library and Circuit-Level Modeling

A power (or energy) modeling tool is integrated as the subclass of energy library. Power is characterized at microarchitecture-level components whose circuit-level designs are estimated from supported tools based on technology-dependent parameters and microarchitectural configurations. For each component of simulated microarchitecture, a model of the energy library estimates per-access dynamic energies of distinct operation types (e.g., read, write, logical switching) and leakage power [11], [17], [19], [25], [26]. Total dynamic energy is calculated by multiplying estimated per-access energies with counters of corresponding access types that can be collected from microarchitecture simulations. For example, to estimate the power dissipation of a register file, read and write access counters of this component are collected from a microarchitecture simulation. Per-access energy of a read or write operation is estimated by a selected circuit-level model. Dynamic power is calculated as the sum of the products of access counters and per-access energies for each access type, divided by timing interval that the counters have been collected during the microarchitectural simulation. Leakage power has exponential dependency on temperature, so it requires a thermal analysis for accurate power modeling. While KitFox supports several popularly used tools in the architecture community, integration is not limited to these tools, and new models can be seamlessly added.

B. Thermal Library and Package-Level Modeling

Temperature modeling tools are encapsulated into thermal library. Temperature is characterized at package level. A processor package is represented as the stack of layers with thermal grids. Each thermal grid cell is expressed as a thermal RC connection. Calculating thermal field over a processor package is equivalent to solving the differential equations of this thermal RC network. Components constructing a processor on a die are organized and represented by floorplans [7], [23]. Each floorplan block represents a set of microarchitectural units, where the power dissipation of each unit is estimated from a linked energy library model and microarchitectural timing simulation. The power estimates of floorplans are supplied to a gridding process that computes power at each grid cell at a finer resolution (configurable within a thermal model). This fine-grained power grid (e.g., 0.01mm^2 per cell) is the input to the thermal model, and temperature field over this grid is calculated. Temperature changes are coupled to leakage power and create a feedback loop between the energy and thermal libraries.

C. Reliability Library and Lifetime Prediction

Device aging phenomena include electro-migration (EM), negative bias temperature instability (NBTI), time-dependent dielectric breakdown (TDDB), hot carrier injection (HCI), and stress migration (SM). Since these failure mechanisms reflect long-term behaviors, they are impractical to simulate across the entire processor at cycle level. Thus, high-level abstractions are generally employed in lifetime reliability studies [24]. In KitFox, detailed modeling of physical interactions is utilized to calculate failure rates and evaluate lifetime reliability. Cumulative failure rates are calculated with respect to time-varying stress conditions including voltage and thermal states that are induced by workload dynamics and microarchitectural operations. Core or processor-level failure rates are calculated as the sum of failure rates of all constituent components. This library is coupled with thermal and energy libraries through physical interactions chain. The architecture-level modeling of lifetime reliability and multi-physics interactions enables us to explore more complex problems such as understanding the reliability criticality of different applications or dynamic execution controls (e.g., turbo-mode executions). For example, if core execution is accelerated by elevating voltage and clock frequency, it increases switching activities of core components and voltage stress. Increased power and heat dissipation accelerates device degradation processes. As a result, the failure rate of the core rises, and predicted lifetime decreases. Such interactions cannot be correctly captured without detailed multi-physics modeling.

IV. Processor Physics: Modeling Application-Microarchitecture-Physics Interactions

We refer to interactions between diverse physical phenomena and their impact on multicore processor performance as *processor physics*. Modeling the processor physics is an important but challenging task. Execution controls for system-level tasks such as power or thermal management create spatiotemporal variations of physical phenomena in multicore processors in addition to that created by executed applications. It is impossible to model such interacting physical phenomena with the first-order analysis (e.g., steady-state or trace-driven simulation), where a single physical property is characterized at a time under oversimplified assumptions of the behaviors of other physical effects. Therefore, integrated cross-layer, multi-physics simulation is an imminent modeling challenge for system-level research explorations across applications, microarchitectures, and diverse physical phenomena.

In KitFox framework, multiple physical models are concurrently simulated, and their interactions are captured at user-defined sampling rate. At every sampling interval, KitFox is invoked by a cycle-level microarchitecture simulator. Figure 2 depicts an architecture-level abstraction of interactions between multiple physical properties and associated models. Execution of workloads in the microarchitecture simulation generates switching activities of functional components (e.g., register file). Architectural activities are represented by access counters (e.g., read, write) and used to calculate the dynamic power dissipation of modeled components [11], [17], [19], [25], [26]. Leakage power is estimated by assuming a constant temperature during a sampling interval. Power results are mapped onto user-created thermal floorplans, and temperature is calculated based on

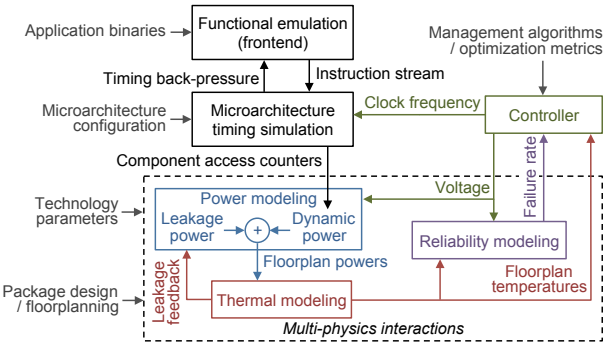


Figure 2. Architecture-level abstraction of interactions between multiple physical properties and microarchitecture. These interactions are captured concurrently during simulation runtime [20], [21].

the spatial distribution of input powers and thermal grid states [7], [23]. Thermal changes cause feedback interactions between temperature and leakage power, and recalculated leakage power is used for temperature calculation in the next sampling period. Reliability characteristics (i.e., failure rate and lifetime prediction) of modeled components are calculated with respect to time-varying operating conditions (i.e., voltage and temperature). The chain of physical interactions creates a loop and is repeated at every sampling interval. Execution controls such as dynamic voltage and frequency scaling (DVFS) may intervene in this chain of events and dynamically change operating conditions and resulting physical phenomena.

V. Data Manipulation and Errors

In KitFox, *data queues* are used to store calculated results from multicore timing simulations and associated physical models. Organization and operation of data queues are central to the correct modeling of multi-physics interactions. Their design, implementation, and operation are described in this section.

A. Data Formats

When integrating multiple physical models (especially third-party tools) in the same framework, the same physical quantity may be expressed in multiple distinct ways across the tools using different notations, data types (e.g., fixed vs floating point), or even units (e.g., V vs mV). There are pragmatic engineering problems to convert between different formats of data shared across multiple tools. KitFox defines the data formats of common physical phenomena (e.g., voltage, power), and the wrapper classes of integrated tools handle data conversions. Therefore, library classes return computed results in consistent data formats that can be shared with other libraries, and data manipulation can be standardized in KitFox.

B. Data Queues and Operations

A challenge in data management is the maintenance of time-varying data that are shared across multiple libraries. For instance, power is calculated by an energy library model and used to compute temperature by a thermal model. Changes in temperature cause feedback interactions with leakage power and update temperature-dependent variables of the energy model. If users or other libraries naively refer to the power data or variables stored in the energy library model, inconsistent results will be returned depending on whether the request is made before or after the calculations or feedback interactions are performed. Thus, time synchronization and retaining calculated results are essential to the correct modeling and calculation of multi-physics interactions. In KitFox, computed results from each physical model are stored in *data queues* to handle cross-reference between libraries and runtime updates of models. Each data queue stores a single type of data (e.g., power) and is identifiable with type information that indicates which type of data is stored within. Each element in the queue is time-stamped with i) simulation time at which it was created and ii) sampling interval at which it was recorded. Since computed results are stored in separate structures rather than overwriting the variables of models, users or other libraries can refer to the correct results with time tags while runtime updates of the models can independently proceed without corrupting the results. There are two types of queues to manage time-varying data as follows.

1) *Closed queue* (for periodic data): Discrete-time data such as power or temperature are calculated and stored at the end of sampling interval, time $= t$, based on observed statistics during period p. These data are regarded as valid during $(t-p,\ t]$ duration, and the queue returns the data for any access requests between $t-p$ and t (sec) including t (sec).
2) *Open queue* (for aperiodic data): Some types of data are aperiodically collected during runtime simulations. For example, clock frequency remains constant until a control mechanism (e.g., DFS) decides to change the frequency. In such a case, datum stored at time $= t$ is valid for $[t, \infty)$ or until a new value is inserted at $t+\delta$ (sec). The queue returns the data for access requests between t and $t+\delta$ (or ∞) including t (sec).

There are three operations defined for data queue accesses; *push* (data insertion), *pull* (data retrieval), and *overwrite* (data replacement). Push and pull are basic write and read operations of the queue, respectively. A pull operation does not dequeue an entry, but dequeueing is implicitly handled when the queue becomes full (at user-defined capacity). An overwrite function replaces an existing entry with a new value. All queue operations require a data identifier (i.e., enumerated type) and tag information (i.e., time t and sampling period p). A queue operation first finds a data structure with the data identifier, where each queue stores a single type of data. Time tag is checked at every queue operation, and error detection is provided for functional correctness and debugging. For a push operation, data intervals must be contiguous, and timing violations set error codes (e.g., non-contiguous, overlapped, out-of-order). If the period p is not provided (i.e., $p = 0$), the queue operation implicitly derives the period value by checking the last entry in the queue. For a pull operation, a data request must match with the time tag (t and p pair) of an existing entry. If no matching entry is found, an error code is set (e.g., tag-mismatch, out-of-queue-range). If the period p is not provided, it tries to find an interval that the time t falls into and returns the data value of that interval. Overwrite is a combined operation of pull and push accesses. It first performs the same process as the pull operation and then replaces a data value if matching entry is found.

C. Data Queues and Library Callbacks

Data queues are coupled with library classes such that inserting new data into the queues (e.g., push operation) triggers the *callback functions* of associated libraries. The callback functions perform updating the variables of library models that are dependent on the inserted data type. For instance, if a microarchitecture simulator needs to perform dynamic voltage scaling (DVS), this is simply inserting a new voltage value into the data queue associated with an energy library model through a KitFox API call. Then, the push operation triggers the callback function of linked energy library, which updates the voltage-dependent variables of subclass power model. As such, runtime updates of library models can be greatly

TABLE II. Error Propagation in Physical Interactions Chain

Power inputs		Error propagation in results	
Power density (avg. temp.)	Max input error	Resulting temp. error	Resulting MTTF error
50 W/cm^2 (68°C)	10%	1.6%	2.6%
	30%	4.9%	7.9%
	50%	7.6%	12.7%
100 W/cm^2 (90°C)	10%	2.4%	3.7%
	30%	6.9%	11.3%
	50%	11.4%	20.2%
125 W/cm^2 (101°C)	10%	2.8%	4.2%
	30%	7.7%	12.4%
	50%	12.6%	21.7%

simplified and automated by managing data in the queues since transferring data between the queues implicitly incur the updates of associated libraries. Specific implementations of callback functions have to be defined by the wrapper classes of individual models.

D. Error Propagation in Physical Interactions Chain

With interacting physical models, inaccuracies in one model can propagate through (and sometimes be amplified by) other models. As described earlier, KitFox does provide the detection of timing and synchronization errors. We view the need for model validation as being important but distinct from the goals of this paper, which is developing a multi-physics simulation environment with interacting models. Moreover, the individual models (i.e., third-party modeling tools) are not the contribution of this paper. As KitFox is designed to incorporate external modeling tools, we are concerned with how inaccuracies can propagate through interacting models and thereby affect simulation results.

To study this issue, we created 8×8 checkerboard floorplans of 256mm^2 area and assumed even power density on the die. Steady-state temperature and resulting mean time to failure (MTTF) of each floorplan were calculated under these conditions. Then, uniformly distributed errors were added to the power inputs, and corresponding changes in resulting temperature and MTTF were measured. We used HotSpot steady-state thermal model with default parameters [7] in this experiment, and wear models presented in the work of Song et al. [21] were used to estimate the MTTF. Table II shows the changes of resulting temperature and MTTF due to injected errors in the power inputs. With increasing power density, the same input error magnitude produces larger absolute changes in power density and hence resulting temperature and MTTF. Due to thermal spreading and low-pass filtering effects, temperature changes are much smaller than the error rates induced in the power inputs. However, non-linearity in MTTF equations is biased against thermal hotspots, and its error increases with greater unevenness in power and thermal densities. In overall, this experiment reveals that errors in power inputs are at least not significantly magnifying through the physical interactions chain.

VI. Processor Component Hierarchy and Description

We seek to devise a composable framework that enables users to simulate various different microarchitectures or package designs. A principal challenge is in interconnecting multiple libraries and configuring their subclass models corresponding to a target processor design to simulate, while being able to flexibly change model parameters or modify processor designs. In KitFox, this is achieved by implementing a unified configuration method to define a processor component hierarchy (e.g., packages, floorplans, microarchitecture components), where each component in the hierarchy is associated with a library model.

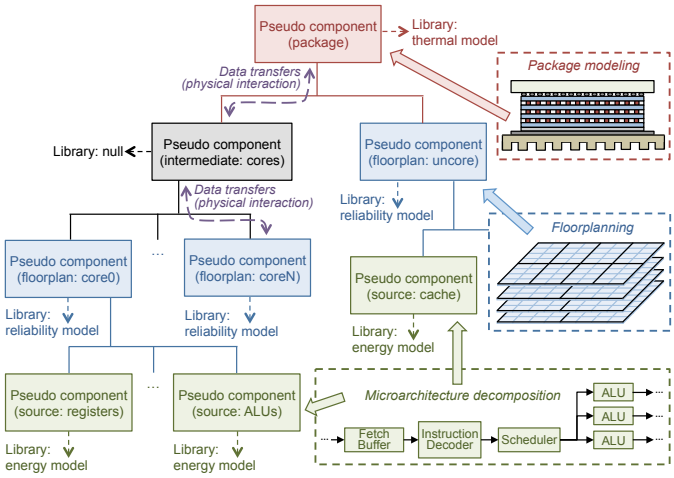


Figure 3. A processor is represented with *pseudo component hierarchy* in KitFox. Pseudo components are physically defined units where associated libraries and their subclass models simulate physical phenomena. Physical interactions are emulated by transferring data between the data queues of pseudo components [20].

A. Representation of Processor Component Hierarchy

In KitFox framework, a processor is represented as the hierarchy of *pseudo components*. Pseudo components are abstract units for which library models are attached to simulate physical phenomena. Different physical properties are characterized at different levels of processor abstraction. For instance, power is characterized at microarchitecture or circuit-level components using activity counts (refer to Section III-A for the energy library and power calculation method). Temperature is calculated at package level based on the power distribution of a processor die. As such, pseudo components can represent different levels of processor abstraction depending on which library models they are associated with and which physical properties are characterized. A pseudo component may represent a microarchitecture component when an energy library model is associated with it to calculate power or energy dissipation. Or it can be a processor package if a thermal library model is attached. A pseudo component hierarchy can be flexibly composed to simulate different processor designs. There are no inherent restrictions on the number of levels in the pseudo component tree, and each pseudo component can have as many sub-components as necessary.

Figure 3 illustrates an example of how KitFox framework serves to interface pseudo components and libraries to simulate a processor design. Simulated microarchitecture is decomposed into basic components (shown as "sources" in the figure), where power is estimated by energy library models. Each energy library may derive a different tool, so it enables choosing the most appropriate model for different microarchitecture components (refer to Section III for the description of libraries or wrapper classes). Pseudo components can be grouped into another upper-level pseudo component (shown as "floorplans" in the figure) depending on their microarchitectural and technological similarities (e.g., core, cache). Higher-level components may represent larger processor units such as cores or regions on the die. The root component in the example represents a processor package and is linked to a thermal library model. It can designate any descendant components in the tree as its constituent floorplans. Some intermediate components without linked libraries can also be created for the convenience of processor description or data collection. Every pseudo component includes data queues to store computed results of library models and shared data (e.g., voltage, clock frequency,

power) for cross-referencing between pseudo components (refer to Section V for data queue operations).

When composing a pseudo component hierarchy, users have to know what the models of choice are capable of simulating and how they are configured. The users should specify the input parameters of each model to be used at the corresponding pseudo component. KitFox itself does not perform microarchitecture-aware automation. Technically, KitFox only recognizes the pseudo component hierarchy and libraries associated with the components. For instance, KitFox does not know if a pseudo component is representing a register file or cache but treats each pseudo component in the tree as a unit linked to one of the libraries. Microarchitecture simulators are responsible for providing complete activity statistics (e.g., access counters, timing information) with KitFox and its library models. This approach tackles a problem that developers write simulators in many different ways. There is no common way to organize the simulation models of microarchitectural blocks into specific C/C++ functions or classes. The notion of pseudo components enables simulator users to map code segments from specific simulators to KitFox library models thereby making it easier to incorporate the multi-physics framework into any existing simulators and to do so in a portable manner.

B. Steps in KitFox Framework Executions and API Functions

KitFox provides a set of API functions for data calculation or manipulation that can be used in user microarchitecture simulators. A pseudo code example is shown in Algorithm 1. Microarchitecture simulation is organized as a sequence of sampling intervals. At the end of every sampling interval, collected access counters are used to calculate the power dissipation of modeled components, and the results are stored in the data queues of corresponding pseudo components. Power data are synchronized in the pseudo component hierarchy by aggregating the values from the leaves toward the root of the tree. The data queues of pseudo components without energy libraries (e.g., floorplans or package) are also updated based on the power values of constituent components. Since data in the queues are tagged with time information, timing violation can be detected when pseudo components have asynchronous power data or if the powers of some components are mistakenly not calculated (refer to Section V for data queue operations). This synchronization process is handled inside KitFox, and any pseudo components can be probed to retrieve the power data after synchronization.

Updated power information of floorplan-level components is used to calculate temperature. A thermal library model internally converts floorplan powers to grid-level power distribution, updates thermal states, and translates grid-level thermal states to floorplan temperatures. When synchronizing temperature data in the pseudo component tree, it is assumed that temperature is uniform within each floorplan component if no further placement information is provided with its sub-components. If there are multiple sub-components belonging to the same floorplan component, they are updated with the same temperature. This process is technically inserting new temperature values with time tags into data queues (i.e., push operation). As a result, the callback functions of library models (e.g., energy library) are invoked, and dependent variables and states are updated (e.g., thermal and leakage power dependency). Since the calculated results are stored in data queues, library models can safely update their internal variables and states based on defined interactions with time-varying physical properties. Lifetime reliability is characterized at floor-level components in the example. Cumulative failure rates are calculated with respect to time-varying stress conditions including voltage and temperature, and resulting MTTF is estimated. Instead of using one representative value (e.g., average temperature) for the entire processor, KitFox utilizes these interacting physical phenomena during microarchitectural simulations for reliability characterization. Although this approach may not give precise prediction for unknown future operations, the likelihood estimation of MTTF can be used to address relative reliability criticality of different microarchitecture operations or applications.

```
1  while (program runs) do
2      /* Access counters of all decomposed components are
       collected during microarchitecture simulations. */
3      do (microarchitecture simulation and counters collection)
4      /* At the end of interval, KitFox API functions are called. */
5      if (at the end of sampling interval) then
6          /* Calculate the power dissipation of all modeled
           microarchitecture components. */
7          for (all microarchitecture components) do
8              kitfox->calculate_power(uarch_component_id,
               current_time, sampling_interval, counters);
9          end
10         /* Temperature is calculated after power calculations are
           done. Data synchronizations are internally performed. */
11         kitfox-> calculate_temperature(pkg_component_id,
           current_time, sampling_interval);
12         /* Failure rates are calculated with the components
           associated reliability library (see Figure 3 illustration). */
13         for (all floorplan components) do
14             kitfox-> calculate_failure_rate(flp_component_id,
               current_time, sampling_interval);
15         end
16         /* Any pseudo components can be probed to retrieve
           data from their queues. */
17         power_t core_power;
18         int err = kitfox-> pull_data(core_component_id,
           current_time, sampling_interval,
           KITFOX_DATA_POWER, &core_power);
19         /* Voltage scaling can be done by inserting a new value
           into the queue and synchronizing pseudo components. */
20         Volt core_voltage = 1.0; // 1.0V
21         int err = kitfox-> push_and_synchronize_data
           (core_component_id, current_time, sampling_interval,
           KITFOX_DATA_VOLTAGE, &core_voltage);
22         /* Reset access counters at the end of interval. */
23         do (reset all microarchitectural access counters)
24     end
25 end
```

Algorithm 1: A pseudo code example of KitFox API functions in a microarchitecture simulation loop.

After data synchronization, any pseudo components can be probed to retrieve data from queues. Dynamic execution controls such as voltage or frequency scaling can be simply applied by inserting new values into the queues and synchronizing the data in the pseudo component hierarchy. Voltage and frequency synchronizations are performed in a similar manner as temperature synchronization; all descendent components are updated with the same voltage and frequency values, and the callback functions of library models are invoked to update dependent variables. For instance, changing the voltage of a core-level component (shown as one of the floorplans in Figure 3) updates all its sub-components within the core to the same voltage. In sum, pseudo components enable flexible composition of processor designs and interconnection of various library models.

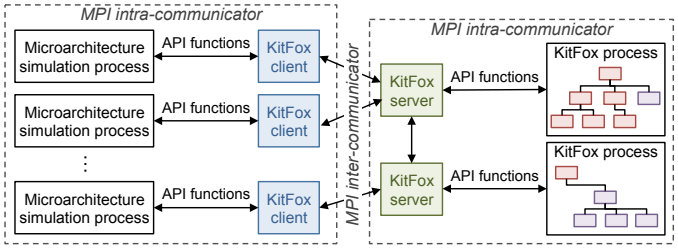
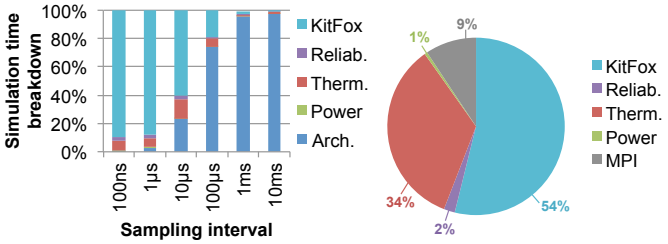


Figure 4. An example of multi-process execution of KitFox framework in parallel with microarchitecture simulation processes via MPI implementations and split communicators.



(a) (b)

Figure 5. (a) Time breakdown of an exemplary multi-physics and microarchitecture simulation with varying sampling intervals. (b) Time breakdown of multi-physics calculations in an 8-process parallel simulation at 1ms sampling rate.

Physical interactions can be easily modeled via data queue operations and callback functions of libraries. Although KitFox supports automated synchronization and timing error checking for easier data manipulation and correct data calculations, it does not implement optimizing system configurations with respect to particular metrics (e.g., energy efficiency). Such optimizations are realized external to the modeling environment (i.e., controller in Figure 2) by utilizing KitFox to extract physical data (e.g., power, thermal) and tune model parameters via API functions.

C. Parallel Interface for Scalable Simulations

Serial simulations can be time consuming when employing computationally intensive physical models over large number of components (e.g., high core-count processors). KitFox framework, developed with SST [16] and Manifold [27] parallel simulators, supports parallel simulation environment via MPI implementations. In particular, KitFox framework can run in parallel with microarchitecture simulators, or KitFox itself can run in multiple MPI processes by dividing pseudo component hierarchy into multiple parts.

In multi-process simulations, each KitFox process initiates a *server* to handle MPI messages from/to other KitFox instances in different processes as well as client microarchitecture simulators. KitFox servers wait for MPI messages to arrive, identify message types, call necessary KitFox API functions, and return the results if necessary. Calculation functions are all non-blocking functions, and microarchitecture simulation can proceed without waiting for the KitFox processes to finish calculations. However, data manipulation has to be blocking since it requires return values (e.g., access to data in queues) for the request. To run KitFox in parallel with an MPI-based microarchitecture simulator, the MPI communicator (i.e., a message channel) has to be split to isolate the communication of parallel microarchitecture simulation from the parallel KitFox processes. Otherwise, the microarchitecture simulator may happen to wait for the MPI barriers of KitFox processes, or vice versa. KitFox servers are fully connected with each other within an MPI *intra-communicator*, and messages from/to client simulators are through an *inter-communicator*. By dividing the pseudo component hierarchy into multiple processes, each KitFox instance only creates and initializes pseudo components that are modeled in its process. The microarchitecture simulator calls KitFox API functions through *client objects* that handle message interfaces to KitFox servers. The KitFox servers remain active until all the simulator processes terminate by sending disconnect messages.

The major difficulty of using parallel simulations is in the correct synchronization of parallel processes. Currently, KitFox runs in a user-driven manner, which requires user microarchitecture simulators to invoke KitFox API functions in a correct time sequence. When API invocations occur out of order in parallel simulations, KitFox detects timing errors and prevents time-incorrect calculations. Synchronization across parallel processes becomes more difficult when microarchitecture components dynamically change operating clock frequencies. A possible improvement will be relaxing the user responsibility for correct thread synchronization in parallel simulations by allowing KitFox to detect the progress of parallel threads. Such parallel simulation optimizations present substantive research challenges in their own right.

D. Simulation Efficacy of Multi-Physics Simulations with KitFox

When multiple models are simultaneously simulated, simulation speed is generally of a concern. Here, we demonstrate that multi-physics simulations are not excessively time-intensive in typical ranges of sampling rates. Figure 5 shows the simulation time breakdown of an exemplary multi-physics simulation with KitFox integrated with Manifold microarchitecture timing simulator [27]. The cycle-level timing simulation of Manifold is known to be as fast as 200 kilo-instructions per second (KIPS) with a single-thread simulation, which is several times faster than highly detailed microarchitecture simulators that are generally known to run around or less than 50 KIPS of simulation speed [13]. We choose McPAT [11] and HotSpot [7] that are the most popular open-source power and thermal modeling tools in the architecture community. In the exemplary simulation, 64 out-of-order cores are configured by adapting the Intel Xeon processor model of McPAT, and the transient thermal model of HotSpot is used with 64×64 grid configuration. The pseudo component hierarchy of KitFox is built similar to Figure 3. Simulation time depends on i) which models are selected, ii) how they are configured (e.g., number of cores), and iii) actual hardware that runs the simulation. Hence, results in Figure 5 are to show the performance of an exemplary multi-physics simulation and do not represent the optimized (or the best) performance that individual simulators or models can achieve. In Figure 5.(a), the interval over which counters of the microarchitecture timing simulator are sampled for calculations is varied between 100ns and 10ms. The clock frequency of simulator components is set to 1.0GHz, so 100ns corresponds to 100 clock cycles in this example. The result shows that the multi-physics simulation is primarily dominated by KitFox operations when the sampling interval is short (less than 10μs), but the microarchitecture simulation becomes the bottleneck when the interval is sufficiently long (greater than 100μs). Here, the power, thermal, or reliability portion in the figure denotes only computation time excluding the handling of input parameters, status updates, or data synchronizations that are all counted as KitFox operations. When these models are individually used, we notice that they also spend significant duration of time on handling input and output data rather than computations, and these parts

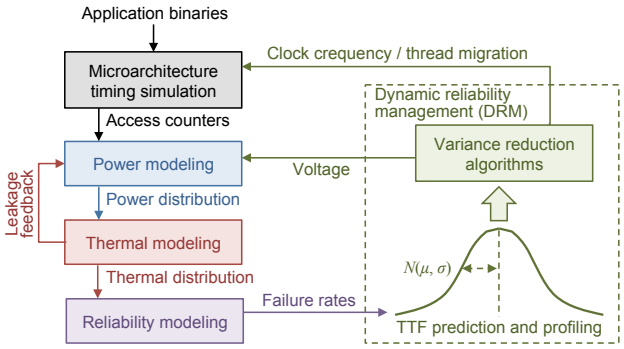


Figure 6. KitFox multi-physics simulation for lifetime reliability characterization and management of multicore processors [21], [22].

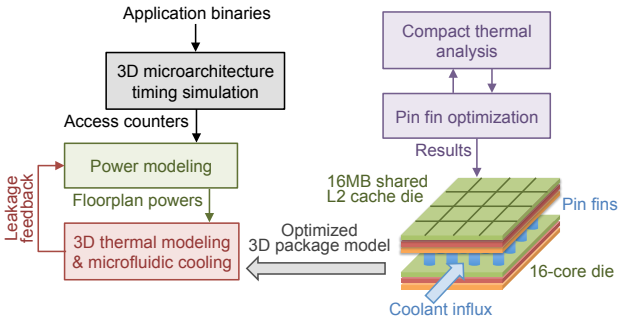


Figure 7. Simulation flow for leakage power characterization and reduction in 3D ICs with microfluidic cooling [29].

are counted as KitFox operations in the integrated multi-physics simulation. In the typical range of sampling intervals (e.g., 100μs or greater), microarchitecture simulation is the bottleneck rather than multi-physics calculations. Therefore, we argue that multi-physics simulation is not as time-intensive as to be the bottleneck, while developing such a multi-physics simulation environment via the integration of various physical models is a highly complicated and challenging task as addressed by KitFox.

Figure 5.(b) plots the time breakdown of multi-physics calculations when the sampling interval is 1ms in an 8-process parallel simulation via MPI. We notice that the transient thermal calculation time of HotSpot is increasing for larger sampling intervals, so the breakdown in Figure 5.(b) appears different from those in (a) with shorter sampling intervals. The result shows that KitFox operations (mostly data manipulation and synchronization) take the most time in multi-physics calculations, and parallelization via MPI also adds non-negligible overhead to the overall simulation time. However, as shown in Figure 5.(a), the simulation bottleneck is in microarchitecture simulations rather than multi-physics calculations for sufficiently long sampling intervals. Therefore, the main reason for parallelization is to speed up microarchitecture simulations, and KitFox should be able to support multi-process simulations.

VII. Case Studies of Integrated Reliability, Thermal, and Power Simulations

In this section, we present several case studies to show the applications of KitFox and motivate the use of this open-source framework for cross-layer simulations and research. The breadth applications described in this section demonstrate the *versatility* of KitFox. We cite several studies that were conducted using KitFox framework, formerly named as Energy Introspector [20], and describe how it was used to drive those studies.

A. Lifetime Reliability Characterization and Management

In a multicore processor, cores experience different levels of degradation depending on power and thermal status induced by workloads, execution controls, etc. As a result, the processor experiences non-uniform degradation on the multicore die, and early failing cores limit the performance and lifetime of the processor.

Song et al. [21], [22] characterized how the parallel execution of workloads created degradation variation on a multicore die and studied how such application-induced variation was translated to processor-level lifetime reliability. In particular, the spatial distribution of degradation on the multicore die was characterized by using a probabilistic model, and the study showed that reducing the variance of degradation distribution was a key to improve processor-level lifetime and more critical than managing the average of the distribution. In this study, KitFox framework integrated with Manifold simulator [27] and Qsim functional emulator [10] provided a full-system microarchitecture and multi-physics simulation environment including power, thermal, and reliability models. A simulation flow is depicted in Figure 6. During microarchitectural timing simulations, Manifold components invoked KitFox API functions to calculate transient power, thermal, and reliability characteristics. The integrated simulation environment using KitFox enabled simultaneous, multi-dimensional explorations across applications, microarchitectures, controllers, and diverse physical phenomena.

B. Leakage Power Reduction in 3D ICs with Microfluidic Cooling

3D stacking of integrated circuits improves performance and energy efficiency by shortening interconnection lengths between processing and memory entities. However, increased power density per unit volume may threaten the thermal stability of the processor. Microfluidic cooling in 3D ICs can provide superior cooling performance over conventional air-cooled packages.

Xiao et al. [29] explored the performance and energy benefits of microfluidic cooling in a 3D-stack processor using KitFox framework as illustrated in Figure 7. They presented an algorithm that minimized junction temperatures under given power budget by optimizing pin fin dimensions including pin diameter, height, spacing and coolant flow rate. Then, 3D-ICE thermal model [23] integrated in KitFox was configured to simulate the optimized 3D package design. McPAT [11] was used for power modeling, and interactions between leakage power and temperature were captured within the KitFox framework. The authors evaluated the performance and energy impacts of the optimized 3D package with microfluidic cooling by simulating PARSEC benchmarks [3] in Manifold microarchitecture simulator [27]. Results showed that the optimized pin fin geometry could reduce leakage power by 20% for tested benchmarks. In this research, KitFox automated the modeling of power-thermal interactions, which helped the authors simulate the optimized 3D package in the full-system simulation environment.

C. Power, Thermal, and Throughput Regulations via Adaptive Gain Controllers in Multicore Processors

Microprocessors are designed to sustain the worst-case operations such as thermal design power (TDP), but applications rarely operate at these limits. From a performance perspective, it is preferred for the system to be designed for average case behaviors (and therefore higher average performance) and adapt to rarely occurring extreme conditions. Rigorous control models can potentially enable such adaptive operations.

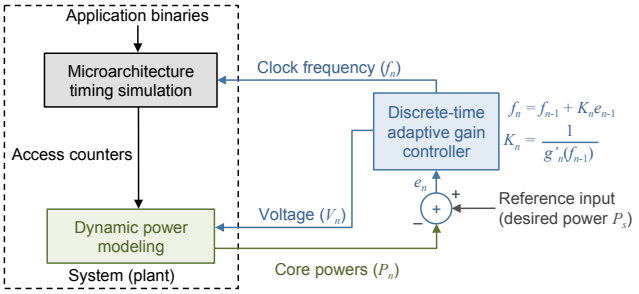


Figure 8. A control system model for processor throughput, power, or thermal regulation via an adaptive gain controller [1], [2], [15].

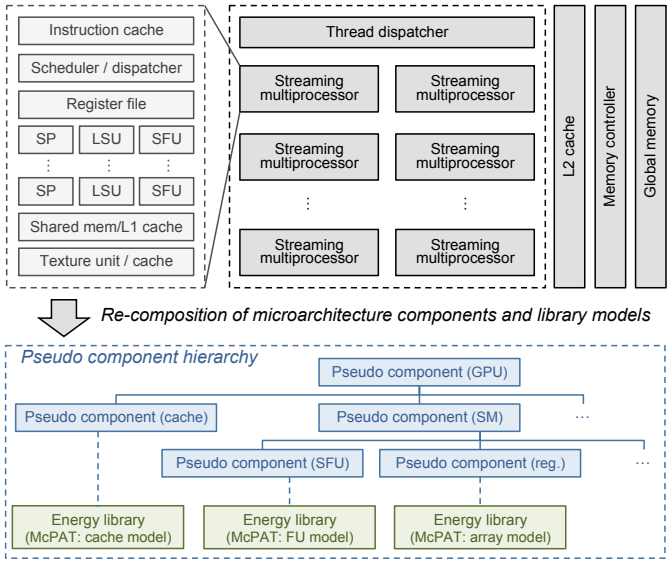


Figure 9. GPU power modeling using KitFox that provides a method to flexibly re-compose microarchitecture components and related power models [12].

Almoosa et al. [1], [2] and Rao et al. [15] presented adaptive gain controller algorithms that regulated throughput (i.e., instructions per second), power, or temperature of multicore processors. The proposed controller algorithms utilized core-level DVFS capability to adjust operating voltages and clock frequencies of cores. A closed-loop system drawn in Figure 8 shows how the system output tracks the input reference based on Newton's method. The authors proposed system models and demonstrated their control algorithms using McPAT [11] and 3D-ICE [23] integrated in KitFox. The DVFS-based control algorithms could be easily applied via KitFox API functions, while these features were not supported in the original modeling tools.

D. Power Modeling of GPU Microarchitectures

Large number of stream multiprocessors (SM) and multiple levels of memory hierarchy in GPUs collectively consume significant amount of power. Power characterization is one of the key challenges in GPU research. Considerable efforts have been invested in the past decades to develop CPU power modeling methods and models, but relatively fewer attempts are found regarding more recent needs for GPU power modeling. Although GPU microarchitectures are substantially different from CPUs, developing a new set of power models involves costly and time-consuming efforts.

Lim et al. [12] approached the GPU power modeling problem by using McPAT [11], a CPU power modeling tool. The authors noted that McPAT was basically comprised of circuit-level models including caches, interconnects, latches, etc., where many of these models could be reused for GPU power modeling. Since McPAT supported only CPU-based microarchitectural designs, the authors used KitFox to re-factor the basic circuit-level models of McPAT to configure an NVIDIA Fermi-like microarchitecture as depicted in Figure 9. Since KitFox interface was independent of specific microarchitecture designs, it could easily adapt to simulate a different microarchitecture while utilizing existing power models.

VIII. Conclusion

Modeling interacting physical phenomena in multicore processors is becoming increasingly important. The proposed KitFox framework facilitates the use of various implementations of physical modeling tools as they are integrated as libraries. The library integration method standardizes interfaces between the models and also enables future extension to other different library types or incorporation of new models. In sum, KitFox makes the following contributions:

- *Standard integration* of physical modeling tools via library interfaces without software inter-dependencies between the tools
- *Coordinated interactions* between the models of multiple distinct physical phenomena
- *User API* for microarchitecture simulators to simplify the use of physical models
- *Composable framework* to simulate different processor designs by flexibly interconnecting the library models
- *Parallel interface* using MPI for scalable simulations

This framework enables multi-dimensional explorations at the intersection of energy, power, temperature, and reliability in conjunction with microarchitecture and application models. Possible improvements to KitFox framework include the following items:

- Optimization of data manipulation and synchronization processes to reduce KitFox overhead
- Easier thread synchronization in parallel simulations
- System or metric optimization modules
- Support of more libraries (e.g., PDN models)

Acknowledgement

This research was supported by Semiconductor Research Corporation (SRC) under the tasks #2084.001 and #2318.001, IBM/SRC Graduate Fellowship, and Sandia National Laboratories.

References

[1] N. Almoosa, W. Song, Y. Wardi, and S. Yalamanchili, "A power capping controller for multicore processors", *American Control Conf.*, pp. 4709-4714, Jun. 2012.

[2] N. Almoosa, W. Song, Y. Wardi, and S. Yalamanchili, "Throughput regulation in multicore processors via IPA", *IEEE Conf. Decision & Control*, pp. 7267-7272, Dec. 2012.

[3] C. Bienia, S. Kumar, and K. Li, "PARSEC vs SPLASH-2: quantitative comparison of two multithreaded benchmark suites on processors", *IEEE Int. Symp. Workload Charact.*, pp. 47-56, Sep. 2008.

[4] A. Bartolini, M. Cacciari, A. Tilli, L. Benini, and M. Gries, "A virtual platform environment for exploring power, thermal, and reliability management control strategies in high-performance multicores", *Great Lakes Symp. VLSI*, pp. 311-316, May 2010.

[5] A. Coskun, T. Rosing, Y. Leblebici, and G. Micheli, "A simulation methodology for reliability analysis in multi-core SoCs", *Great Lakes Symp. VLSI*, pp. 95-99, May 2006.

[6] M. Hsieh, R. Riesen, K. Thompson, W. Song, and A. Rodrigues, "SST: a scalable parallel framework for architecture-level performance, power, area, and thermal simulation", *Comput. J.*, pp. 181-191, Jul. 2011.

[7] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. Stan, "HotSpot: a compact thermal modeling methodology for early-stage VLSI design", *IEEE Trans. VLSI*, vol. 14, no. 5, pp. 501-513, May 2006.

[8] "Failure mechanisms and models for semiconductor devices", JEDEC Solid State Technology Association, JEDEC Publ. JEP122C, Mar. 2006.

[9] A. Kahng, B. Li, L. Peh, and K. Samadi, "ORION 2.0: a fast and accurate NoC power and area model for early-stage design space exploration", *Design, Automat. & Test Europe Conf. & Exhibit.*, pp. 423-428, Apr. 2009.

[10] C. Kersey, A. Rodrigues, and S. Yalamanchili, "A universal parallel frontend for execution driven microarchitecture simulation", *Workshop Rapid Simul. Perform. Eval.*, pp. 25-32, Jan. 2012.

[11] S. Li, J. Ahn, R. Strong, J. Brockman, D. Tullsen, and N. Jouppi, "McPAT: integrated power, area, timing modeling framework for multicore architectures", *IEEE Int. Symp. on Microarchit.*, pp. 469-480, Dec. 2009.

[12] J. Lim, N. Lakshminarayana, H. Kim, W. Song, S. Yalamanchili, and W. Sung, "Power Modeling for GPU Architectures using McPAT", *ACM Trans. Design Automat. Electron. Syst.*, vol. 19, no. 3, pp. 26:1-24, Jun. 2014.

[13] G. Loh, S. Subramaniam, and Y. Xie, "Zesto: a cycle-level simulator for highly detailed microarchitecture exploration", *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, pp. 53-64, Apr. 2009.

[14] S. Priyadarshi, W. Rhett Davis, M. Steer, and P. Franzon, "Thermal pathfinding for 3D ICs," *IEEE Trans. Compon. Packag. Manuf. Technol.*, vol 4, no. 7, pp. 1159-1168, May 2014.

[15] K. Rao, W. Song, S. Yalamanchili, and Y. Wardi, "Temperature regulation in multicore processors using adaptive-gain integral controllers,", *IEEE Conf. Control Appl.*, Nov. 2015.

[16] A. Rodrigues, K. Hemmert, B. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Riesen, J. Cook, P. Rosenfield, E. Cooper-Balis, and B. Jacob, "The structural simulation toolkit", *Int. Workshop Perform. Model. Benchmark. Simul. High Perform. Comput. Syst.*, pp. 37-42, Mar. 2011.

[17] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMsim2: a cycle accurate memory system simulator", *IEEE Comput. Archit. Lett.*, vol. 10, no. 1, pp. 16-19, Jan. 2011.

[18] H. Sajjadi-Kia and C. Ababei, "A new reliability evaluation methodology with application to lifetime-oriented circuit design", *IEEE Trans. Device Mater. Reliab.*, vol. 13, no. 1, pp. 192-202, Mar. 2013.

[19] D. Sekar, A. Naeemi, R. Sarvari, J. Davis, and J. Meindl, "IntSim: a CAD tool for optimization of multi-level interconnect network", *IEEE Int. Conf. Comput.-Aided Design*, pp. 560-567, Nov. 2007.

[20] W. Song, S. Mukhopadhyay, and S. Yalamanchili, "Energy Introspector: parallel, composable framework for integrated power-reliability-thermal modeling for multicore architectures", *IEEE Int. Symp. Perform. Anal. Syst. Softw.*, pp. 143-144, Mar. 2014.

[21] W. Song, S. Mukhopadhyay, and S. Yalamanchili, "Architectural reliability: lifetime reliability characterization and management of many-core processors", *IEEE Comput. Archit. Lett.*, Jul. 2014.

[22] W. Song, S. Mukhopadhyay, and S. Yalamanchili, "Managing performance-reliability tradeoffs in multicore processors,", *IEEE Int. Reliab. Physics Symp.*, pp. 3C.1.1-7, Apr. 2015.

[23] A. Sridhar, A. Vincenzi, M. Ruggiero, T. Brunschwiler, and D. Atienza, "3D-ICE: fast compact transient thermal modeling for 3D ICs with inter-tier liquid cooling", *IEEE Int. Conf. Comput.-Aided Design*, pp. 463-470, Nov. 2010.

[24] J. Srinivasan, S. Adve, P. Bose, and J. Rivers, "Lifetime reliability: toward an architectural solution", *IEEE Micro*, vol. 25, no. 3, pp. 70-80, May 2005.

[25] C. Sun, C. Chen, G. Kurian, L. Wei, J. Miller, A. Agarwal, L. Peh, and V. Stojanovic, "DSENT - a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling", *IEEE Int. Symp. Network Chip*, pp. 201-210, May 2012.

[26] S. Thoziyoor, J. Ahn, M. Monchiero, J. Brockton, and N. Jouppi, "A comprehensive memory modeling tool and its application to the design and analysis of future memory hierarchies", *Int. Symp. Comput. Archit.*, pp. 51-62, Jun. 2008.

[27] J. Wang, J. Beu, R. Bheda, T. Conte, Z. Dong, C. Kersey, M. Rasquinha, G. Riley, W. Song, H. Xiao, P. Xu, and S. Yalamanchili, "Manifold: a parallel simulation framework for multicore systems", *IEEE Int. Symp. Perform. Anal. Syst. Softw.*, pp. 106-115, Mar. 2014.

[28] M. White and J. Bernstein, "Microelectronics reliability: physics-of-failure based modeling and lifetime evaluation", NASA JPL Publ. 08-5. 2008.

[29] H. Xiao, Z. Wan, S. Yalamanchili, and Y. Joshi, "Leakage power characterization and minimization in 3D stacked multi-core chips with microfluidic cooling", *Semicond. Therm. Meas. Manage. Symp.*, pp. 207-212, Mar. 2014.

William J. Song is a Ph.D. candidate at School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA. He received his B.S. degree from School of Electrical Engineering, Yonsei University, Seoul, Korea. He has been an IBM/SRC Graduate Fellow since 2012. He had research internship at Sandia National Laboratories in Albuquerque, NM (2010, 2011, 2012), AMD Research in Bellevue, WA (2013), IBM T.J. Watson Research Center in Yorktown Heights, NY (2014), and Qualcomm in San Diego, CA (2015). He received a Best Student Paper Award at IEEE Reliability Physics Symposium (IRPS) 2015 and Best in Session Awards at SRC TECHCON 2013 and 2014.

Saibal Mukhopadhyay received his B.E. degree in Electronics and Telecommunication Engineering from Jadavpur University, Calcutta, India, in 2000. He received a Ph.D. degree in Electrical and Computer Engineering from Purdue University, West Lafayette, IN, in 2006. He was with the IBM T.J. Watson Research Center, Yorktown Heights, NY, as a Research Staff Member. Since September 2007, he has been with the School of Electrical and Computer Engineering at the Georgia Institute of Technology, Atlanta, GA, where he is currently an Associate Professor. His current research interests include neuromorphic computing, low-power digital and mixed-signal systems, voltage regulation, and power and thermal management. Dr. Mukhopadhyay received the Office of Naval Research Young Investigator Award in 2012, the National Science Foundation CAREER Award in 2011, the IBM Faculty Partnership Award in 2009 and 2010, the SRC Inventor Recognition Award in 2008, the SRC Technical Excellence Award in 2005, and the IBM PhD Fellowship Award for years 2004-2005. He has received the IEEE Transactions on VLSI Systems (TVLSI) Best Paper Award in 2014, the IEEE Transactions on Component, Packaging, and Manufacturing Technology (TCPMT) Best Paper Award in 2014, the IEEE/ACM International Symposium on Low-power Electronic Design (ISLPED) Best Paper Award in 2014 and 2015, the International Conference on Computer Design (ICCD) Best Paper Award in 2004, the IEEE Nano Best Student Paper Award in 2003, and multiple Best in Session Awards in SRC TECHCON in 2014 and 2005. He has authored or co-authored over 150 papers in refereed journals and conferences, and has been awarded six U.S. patents. He is a Senior Member of IEEE.

Sudhakar Yalamanchili earned his Ph.D. degree in Electrical and Computer Engineering 1984 from the University of Texas at Austin. After graduation, he joined Honeywell's Systems and Research Center in Minneapolis where he worked as a Senior, and then Principal Research Scientist from 1984 to 1989. During that time he served as an Adjunct Faculty and taught in the Department of Electrical Engineering at the University of Minnesota, and served on Honeywell's Program Technical Advisory Committee to the Microelectronics Technology Corporation (MSS). He joined the ECE faculty at Georgia Tech in 1989 where he is now a Joseph M. Pettit Professor of Computer Engineering. He research interests lay at the intersection of high performance computing, and power and thermal management techniques, and modeling and simulation of parallel architectures and systems. He served as a Co-Director of the NSF Industry University Research Center on Experimental Computer Systems (CERCS) (2003-2013) and as a member of the Research Advisory Group to the HyperTransport Consortium (2007-2012). He is the author of VHDL Starters Guide, 2nd edition, Prentice Hall 2004, VHDL: From Simulation to Synthesis, Prentice Hall, 2000, and co-author with J. Duato and L. Ni, of Interconnection Networks: An Engineering Approach, Morgan Kaufman, 2003. Dr. Yalamanchili contributes professionally with regular service on editorial boards and conference program committees. He has served on the editorial boards of the IEEE Transactions on Parallel and Distributed Processing and IEEE Transactions on Computers and IEEE Computer Architecture Letters. He also contributes professionally through service on conference and workshop program committees in the area of high performance computing and computer architecture. He is a Fellow of the IEEE.